RF

In [1]:

```
# Load the Drive helper and mount
from google.colab import drive

# This will prompt for authorization.
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

In [0]:

```
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

# using the SQLite Table to read data.
con = sqlite3.connect('database.sqlite')

#filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 """, con)

# Give reviews with Score>3 a positive rating, and reviews with a score<3 a negative rating.
def partition(x):
    if x < 3:
        return 0
    return 1

#changing reviews with score less than 3 to be positive and vice-versa
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
print(filtered_data.shape)
```

In [0]:

```
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='qui
cksort', na_position='last')
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inpl
ace=False)
```

```
final.shape
```

In [0]:

```
final.head(2)
```

In [0]:

```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
#Before starting the next phase of preprocessing lets see the number of entries left
print(final.shape)

#How many positive and negative reviews are present in our dataset?
final['Score'].value_counts()
```

In [0]:

```
final.sort_values('Time',axis=0,ascending=True,inplace=True,kind='quicksort')
final.head(2)
```

In [0]:

```
stop = set(stopwords.words('english')) #set of stopwords
sno = nltk.stem.SnowballStemmer('english') #initialising the snowball stemmer

def cleanhtml(sentence): #function to clean the word of any html-tags
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', sentence)
    return cleantext
def cleanpunc(sentence): #function to clean the word of any punctuation or special characters
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned
```

In [0]:

```
#Code for implementing step-by-step the checks mentioned in the pre-processing phase
# this code takes a while to run as it needs to run on 500k sentences.
i=0
str1=' '
final_string=[]
all_positive_words=[] # store words from +ve reviews here
all_negative_words=[] # store words from -ve reviews here.
s=''
for sent in final['Text'].values:
    filtered_sentence=[]
    #print(sent);
    sent=cleanhtml(sent) # remove HTMl tags
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if((cleaned_words.isalpha()) & (len(cleaned_words)>2)):
                if(cleaned_words.lower() not in stop):
                    s=(sno.stem(cleaned_words.lower())).encode('utf8')
                    filtered_sentence.append(s)
                    if (final['Score'].values)[i] == 1:
                        all_positive_words.append(s) #list of all words used to describe positive r
eviews
                    if(final['Score'].values)[i] == 0:
                        all_negative_words.append(s) #list of all words used to describe negative r
eviews reviews
                else:
                    continue
            else:
                continue
    #print(filtered_sentence)
    str1 = b" ".join(filtered_sentence) #final string of cleaned words
    #print("***********************************************************************")

    final_string.append(str1)
    i+=1
```

In [0]:

```
final['CleanedText']=final_string #adding a column of CleanedText which displays the data after pr
e-processing of the review
final['CleanedText']=final['CleanedText'].str.decode("utf-8")
final.head(3)
```

In [0]:

```
import pickle
pickle.dump(final, open('final.p', 'wb'))
#final_sent = pickle.load(open('data.p','rb'))
final.shape
```

In [2]:

```
import pickle
final = pickle.load(open('drive/My Drive/Colab Notebooks/Random forest/final.p','rb'))


from sklearn.model_selection import train_test_split
##Sorting data according to Time in ascending order for Time Based Splitting
time_sorted_data = final.sort_values('Time', axis=0, ascending=True, inplace=False, kind='quicksort
', na_position='last')
final.head(2)
```

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summ
138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0	1	939340800	EV bo educati
138683	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	2	1	940809600	This w seri great to sp time

In [32]:

```
y = final['Score'].iloc[:100000]
x = final['CleanedText'].iloc[:100000]
x.shape,y.shape
```

Out[32]:

```
((100000,), (100000,))
```

BOW

In [33]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

X_tra, X_tes, y_train, y_test = train_test_split(x,y,test_size=0.3,random_state=0)

#Implementing BAG of words
bow = CountVectorizer(ngram_range=(0,1))
X_tf =bow.fit_transform(X_tra)

# Standerdising the data
norm = StandardScaler(with_mean = False)
```

```
X_train = norm.fit_transform(X_tf)

# tfidf test
X_tfte = bow.transform(X_tes)

# Standerdising the data
X_test = norm.transform(X_tfte)



X_train.shape,y_train.shape,X_test.shape,y_test.shape
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/validation.py:475: DataConversionWarning: Dat
a with input dtype int64 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

Out[33]:

```
((70000, 31572), (70000,), (30000, 31572), (30000,))
```

In [0]:

```

```

In [34]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import make_scorer
from sklearn.metrics import f1_score
from sklearn.model_selection import RandomizedSearchCV

tscv = TimeSeriesSplit(n_splits=3)
param_grid = {'n_estimators':[60,80,100]}
rf=RandomForestClassifier(oob_score=True,class_weight='balanced')
gsv = GridSearchCV(rf,param_grid,cv=tscv,scoring="f1_weighted",n_jobs=-1,pre_dispatch=2)
gsv.fit(X_train,y_train)

print("Best HyperParameter: ",gsv.best_params_)
print("Best f1: %.2f%%"%(gsv.best_score_*100))
```

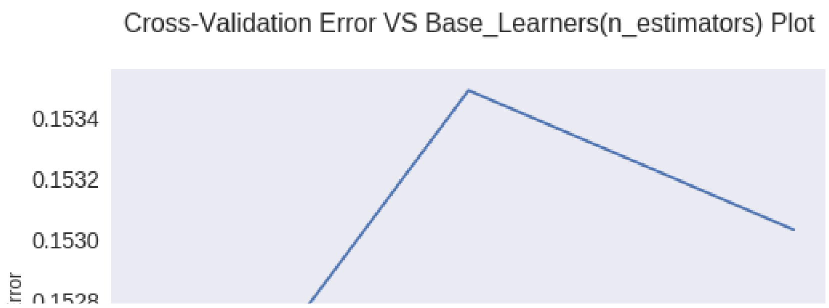
```
Best HyperParameter:  {'n_estimators': 60}
Best f1: 84.79%
```

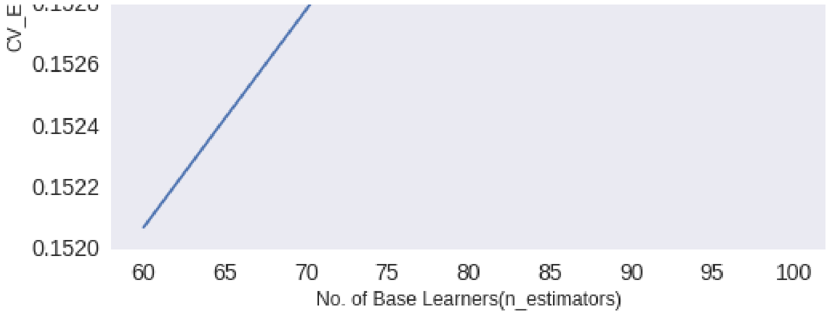
In [0]:

```
cv_errors = [1-i for i in gsv.cv_results_['mean_test_score']]
```

In [38]:

```
base_learners=[60,80,100]
# plotting99 Cross-Validation Error vs Base learners graph
plt.plot(base_learners, cv_errors)
plt.xlabel('No. of Base Learners(n_estimators)',size=12)
plt.ylabel('CV_Error',size=12)
plt.title('Cross-Validation Error VS Base_Learners(n_estimators) Plot\n',size=16)
plt.grid()
plt.show()
```





In [8]:

```
X_train.shape,y_train.shape,X_test.shape,y_test.shape
```

Out[8]:

```
((70000, 31572), (70000,), (30000, 31572), (30000,))
```

In [0]:

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=60,class_weight='balanced')
model.fit(X_train,y_train)
y_pred = model.predict(X_train)
```

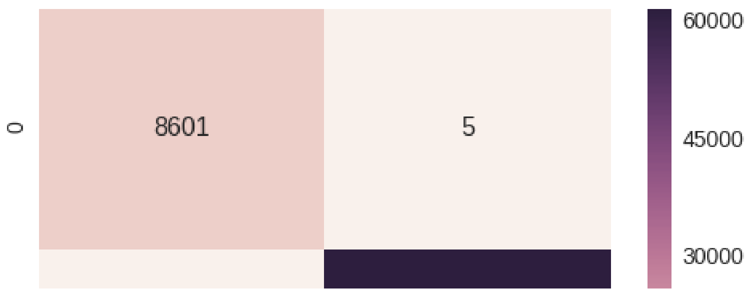
In [10]:

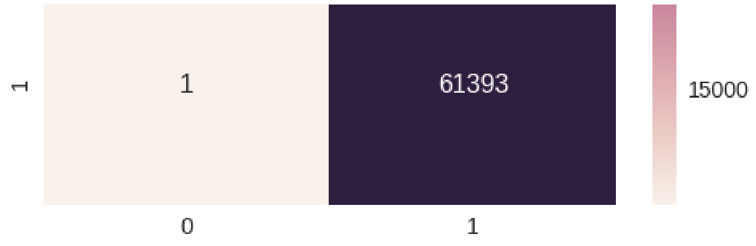
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import seaborn as sns

print("Accuracy on the set: %0.3f%%"%(accuracy_score(y_train, y_pred)*100))
print("F1-Score on the set: %0.3f%%"%(f1_score(y_train, y_pred,average='weighted')*100))
print("Precision_score on test set: %0.3f%%"%(precision_score(y_train, y_pred)*100))
print("Recall_score on test set: %0.3f%%"%(recall_score(y_train, y_pred)*100))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
result = confusion_matrix(y_train,y_pred)
print(result)
sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Accuracy on the set: 99.991%
F1-Score on the set: 99.991%
Precision_score on test set: 99.992%
Recall_score on test set: 99.998%
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[ 8601     5]
 [    1 61393]]
```

Out[10]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4baa6ff128>
```





In [0]:

```
X_test.shape
```

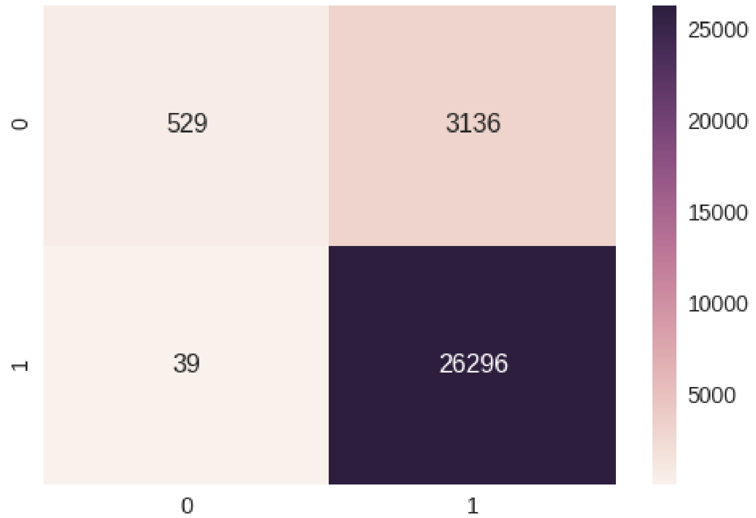
In [11]:

```
y_pred1 = (model.predict(X_test))
print("Accuracy on the set: %0.3f%%"%(accuracy_score(y_test, y_pred1)*100))
print("F1-Score on the set: %0.3f%%"%(f1_score(y_test, y_pred1,average='weighted')*100))
print("Precision_score on test set: %0.3f%%"%(precision_score(y_test, y_pred1)*100))
print("Recall_score on test set: %0.3f%%"%(recall_score(y_test, y_pred1)*100))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
result = confusion_matrix(y_test,y_pred1)
print(result)
sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Accuracy on the set: 89.417%
F1-Score on the set: 85.839%
Precision_score on test set: 89.345%
Recall_score on test set: 99.852%
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[  529  3136]
 [   39 26296]]
```

Out[11]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4baa5982e8>
```



In [0]:

```

```

TFIDF

In [3]:

```
y = final['Score'].iloc[:100000]
```

```
x = final['CleanedText'].iloc[:100000]
x.shape,y.shape
```

Out[3]:

```
((100000,), (100000,))
```

In [0]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

X_tra, X_tes, y_train, y_test = train_test_split(x,y,test_size=0.3,shuffle=False)

#Implementing BAG of words
tfidf = TfidfVectorizer(ngram_range=(0,1),dtype=float)
X_tf =tfidf.fit_transform(X_tra)

# Standerdising the data
norm = StandardScaler(with_mean = False)
X_train = norm.fit_transform(X_tf)

# tfidf test
X_tfte = tfidf.transform(X_tes)

# Standerdising the data
X_test = norm.transform(X_tfte)


from sklearn.model_selection import TimeSeriesSplit
```

In [12]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import make_scorer
from sklearn.metrics import f1_score
from sklearn.model_selection import RandomizedSearchCV

tscv = TimeSeriesSplit(n_splits=3)
param_grid = {'n_estimators':[60,120,180]}
rf=RandomForestClassifier(oob_score=True,n_jobs=-1,class_weight='balanced')
gsv = GridSearchCV(rf,param_grid,cv=tscv,scoring="f1_weighted",n_jobs=-1,pre_dispatch=2)
gsv.fit(X_train,y_train)

print("Best HyperParameter: ",gsv.best_params_)
print("Best f1: %.2f%%"%(gsv.best_score_*100))
```

```
Best HyperParameter:  {'n_estimators': 60}
Best f1: 84.71%
```

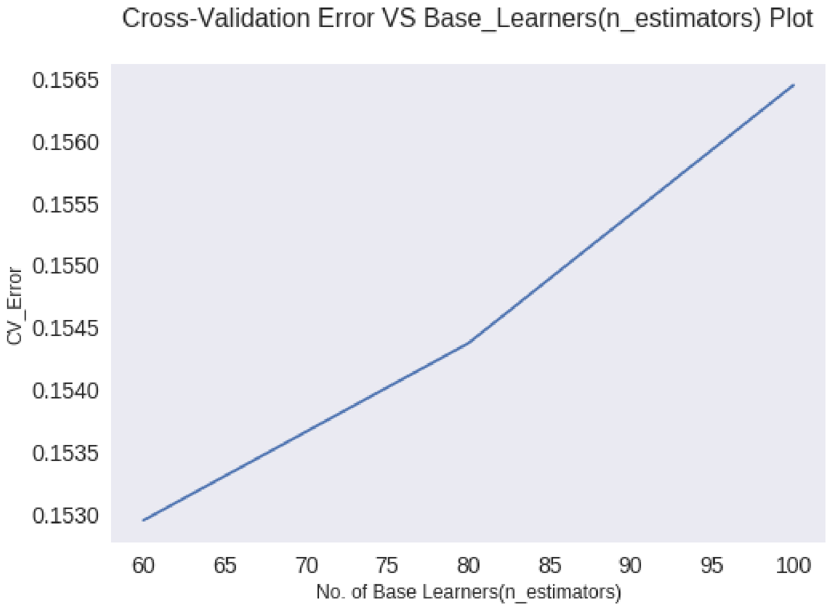
In [0]:

```
#cross-val error
cv_errors = [1-i for i in gsv.cv_results_['mean_test_score']]
```

In [15]:

```
import matplotlib.pyplot as plt
base= [60,80,100]

# plotting Cross-Validation Error vs Base learners graph
plt.plot(base, cv_errors)
plt.xlabel('No. of Base Learners(n_estimators)',size=12)
plt.ylabel('CV_Error',size=12)
plt.title('Cross-Validation Error VS Base_Learners(n_estimators) Plot\n',size=16)
```

```
plt.grid()
plt.show()
```



In [16]:

```
model2 = RandomForestClassifier(n_estimators=60,oob_score=True,n_jobs=-1,class_weight='balanced')
model2.fit(X_train,y_train)
y_pred2 = model2.predict(X_train)
model2
```

Out[16]:

```
RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=None, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=60, n_jobs=-1, oob_score=True, random_state=None,
            verbose=0, warm_start=False)
```

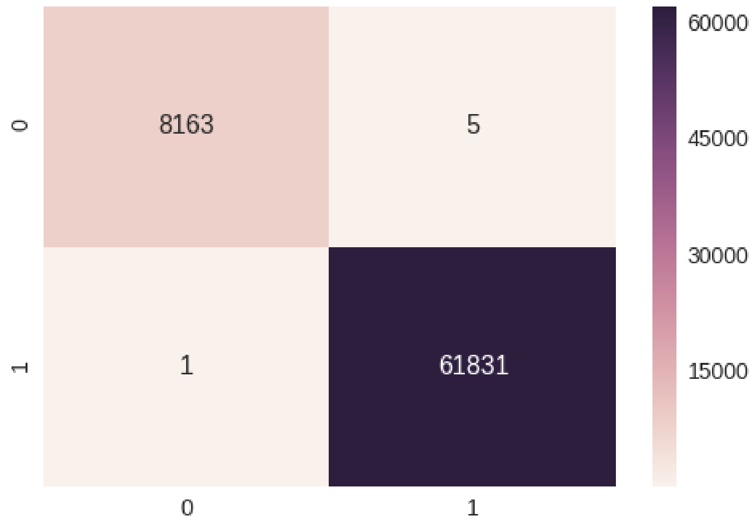
In [17]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import seaborn as sns

print("Accuracy on the set: %0.3f%%"%(accuracy_score(y_train, y_pred2)*100))
print("F1-Score on the set: %0.3f%%"%(f1_score(y_train, y_pred2,average='weighted')*100))
print("Precision_score on test set: %0.3f%%"%(precision_score(y_train, y_pred2)*100))
print("Recall_score on test set: %0.3f%%"%(recall_score(y_train, y_pred2)*100))
result = confusion_matrix(y_train,y_pred2)
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print(result)
sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Accuracy on the set: 99.991%
F1-Score on the set: 99.991%
Precision_score on test set: 99.992%
Recall_score on test set: 99.998%
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[ 8163     5]
 [    1 61831]]
```

Out[17]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f8da81bba90>



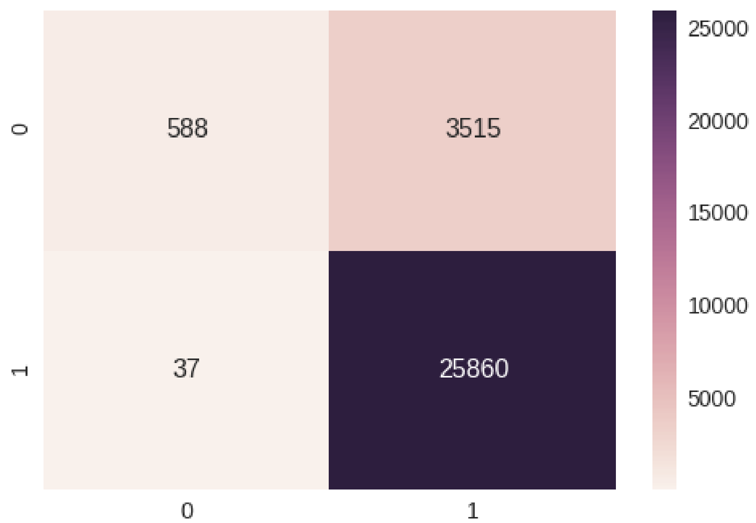
In [19]:

```
y_pred2 = model2.predict(X_test)
print("Accuracy on the set: %0.3f%%"%(accuracy_score(y_test, y_pred2)*100))
print("F1-Score on the set: %0.3f%%"%(f1_score(y_test, y_pred2,average='weighted')*100))
print("Precision_score on test set: %0.3f%%"%(precision_score(y_test, y_pred2)*100))
print("Recall_score on test set: %0.3f%%"%(recall_score(y_test, y_pred2)*100))
result = confusion_matrix(y_test,y_pred2)
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print(result)
sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Accuracy on the set: 88.160%
F1-Score on the set: 84.178%
Precision_score on test set: 88.034%
Recall_score on test set: 99.857%
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[  588  3515]
 [   37 25860]]
```

Out[19]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f8da667e908>



Avg w2c

In [44]:

```
!pip install gensim
```

```
Collecting gensim
  Downloading
https://files.pythonhosted.org/packages/27/a4/d10c0acc8528d838cda5eede0ee9c784caa598dbf40bd0911ff8c
7eb/gensim-3.6.0-cp36-cp36m-manylinux1_x86_64.whl (23.6MB)
    100% |████████████████████████████████| 23.6MB 1.7MB/s
Requirement already satisfied: six>=1.5.0 in /usr/local/lib/python3.6/dist-packages (from gensim)
(1.11.0)
Requirement already satisfied: numpy>=1.11.3 in /usr/local/lib/python3.6/dist-packages (from
gensim) (1.14.6)
Requirement already satisfied: scipy>=0.18.1 in /usr/local/lib/python3.6/dist-packages (from
gensim) (0.19.1)
Collecting smart-open>=1.2.1 (from gensim)
  Downloading
https://files.pythonhosted.org/packages/4b/1f/6f27e3682124de63ac97a0a5876da6186de6c19410feab66c1543
055/smart_open-1.7.1.tar.gz
Collecting boto>=2.32 (from smart-open>=1.2.1->gensim)
  Downloading
https://files.pythonhosted.org/packages/23/10/c0b78c27298029e4454a472a1919bde20cb182dab1662cec7f2ca
523/boto-2.49.0-py2.py3-none-any.whl (1.4MB)
    100% |████████████████████████████████| 1.4MB 14.6MB/s
Collecting bz2file (from smart-open>=1.2.1->gensim)
  Downloading
https://files.pythonhosted.org/packages/61/39/122222b5e85cd41c391b68a99ee296584b2a2d1d233e7ee32b453
f2d/bz2file-0.98.tar.gz
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-packages (from smart-
open>=1.2.1->gensim) (2.18.4)
Collecting boto3 (from smart-open>=1.2.1->gensim)
  Downloading
https://files.pythonhosted.org/packages/94/04/c48c102e11b0cb2e3d4a7bdda49647b40e2ae03279ce9ba935e4a
b89/boto3-1.9.34-py2.py3-none-any.whl (128kB)
    100% |████████████████████████████████| 133kB 29.8MB/s
Requirement already satisfied: idna<2.7,>=2.5 in /usr/local/lib/python3.6/dist-packages (from
requests->smart-open>=1.2.1->gensim) (2.6)
Requirement already satisfied: urllib3<1.23,>=1.21.1 in /usr/local/lib/python3.6/dist-packages
(from requests->smart-open>=1.2.1->gensim) (1.22)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/python3.6/dist-packages
(from requests->smart-open>=1.2.1->gensim) (3.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packages (from
requests->smart-open>=1.2.1->gensim) (2018.10.15)
Collecting botocore<1.13.0,>=1.12.34 (from boto3->smart-open>=1.2.1->gensim)
  Downloading
https://files.pythonhosted.org/packages/91/83/3185727fb3d0204bc2d09ebfbf26bf6725e75f70f35cda477f9b9
61d/botocore-1.12.34-py2.py3-none-any.whl (4.7MB)
    100% |████████████████████████████████| 4.7MB 7.0MB/s
Collecting s3transfer<0.2.0,>=0.1.10 (from boto3->smart-open>=1.2.1->gensim)
  Downloading
https://files.pythonhosted.org/packages/d7/14/2a0004d487464d120c9fb85313a75cd3d71a7506955be458eebfe
b1d/s3transfer-0.1.13-py2.py3-none-any.whl (59kB)
    100% |████████████████████████████████| 61kB 24.0MB/s
Collecting jmespath<1.0.0,>=0.7.1 (from boto3->smart-open>=1.2.1->gensim)
  Downloading
https://files.pythonhosted.org/packages/b7/31/05c8d001f7f87f0f07289a5fc0fc3832e9a57f2dbd4d3b0fee70e
365/jmespath-0.9.3-py2.py3-none-any.whl
Collecting docutils>=0.10 (from botocore<1.13.0,>=1.12.34->boto3->smart-open>=1.2.1->gensim)
  Downloading
https://files.pythonhosted.org/packages/36/fa/08e9e6e0e3cbd1d362c3bbee8d01d0aedb2155c4ac112b19ef3ca
d8d/docutils-0.14-py3-none-any.whl (543kB)
    100% |████████████████████████████████| 552kB 24.5MB/s
Requirement already satisfied: python-dateutil<3.0.0,>=2.1; python_version >= "2.7" in
/usr/local/lib/python3.6/dist-packages (from botocore<1.13.0,>=1.12.34->boto3->smart-open>=1.2.1->
gensim) (2.5.3)
Building wheels for collected packages: smart-open, bz2file
  Running setup.py bdist_wheel for smart-open ... -   done
  Stored in directory:
/root/.cache/pip/wheels/23/00/44/e5b939f7a80c04e32297dbd6d96fa3065af89ecf57e2b5f89f
  Running setup.py bdist_wheel for bz2file ... -   done
  Stored in directory:
/root/.cache/pip/wheels/81/75/d6/e1317bf09bf1af5a30befc2a007869fa6e1f516b8f7c591cb9
Successfully built smart-open bz2file
Installing collected packages: boto, bz2file, docutils, jmespath, botocore, s3transfer, boto3, sma
```

```
rt-open, gensim
Successfully installed boto-2.49.0 boto3-1.9.34 botocore-1.12.34 bz2file-0.98 docutils-0.14 gensim
-3.6.0 jmespath-0.9.3 s3transfer-0.1.13 smart-open-1.7.1
```

In [0]:

```
X_tra, X_tes, y_train, y_test = train_test_split(x,y,test_size=0.3,shuffle=False)

sent_of_train=[]
for sent in X_tra:
    sent_of_train.append(sent.split())

sent_of_test=[]
for sent in X_tes:
    sent_of_test.append(sent.split())
```

In [46]:

```
#word to vector
from gensim.models import Word2Vec
w2v_model=Word2Vec(sent_of_train,min_count=3,size=200, workers=4)# words which occurs 3 times; 500
dimensions

w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 3 times ",len(w2v_words))
```

```
number of words that occured minimum 3 times  14050
```

In [47]:

```
# compute average word2vec for each review for X_train .
from tqdm import tqdm
import numpy as np



train_vectors = []
for sent in tqdm(sent_of_test):
    sent_vec = np.zeros(200)
    cnt_words =0;
    for word in sent:
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    train_vectors.append(sent_vec)
```

```
100%|██████████| 30000/30000 [00:44<00:00, 669.38it/s]
```

In [48]:

```
train_vectors1 = []
for sent in tqdm(sent_of_train):
    sent_vec = np.zeros(200)
    cnt_words =0;
    for word in sent:
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    train_vectors1.append(sent_vec)
```

```
100%|██████████| 70000/70000 [01:38<00:00, 711.51it/s]
```

In [49]:

```
len(train_vectors),len(train_vectors1)
```

Out[49]:

```
(30000, 70000)
```

In [50]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import TimeSeriesSplit

# Data-preprocessing: Standardizing the data
sc = StandardScaler(with_mean = False)
X_train3 = sc.fit_transform(train_vectors1)
X_test3 = sc.transform(train_vectors)

tscv = TimeSeriesSplit(n_splits=4)


y_train.shape,X_train3.shape,X_test3.shape
```

Out[50]:

```
((70000,), (70000, 200), (30000, 200))
```

In [0]:

```
pickle.dump(X_train3, open('X_train3.p', 'wb'))
pickle.dump(X_test3, open('X_test3.p', 'wb'))
```

In [0]:

```
##############################
X_train3 = pickle.load(open('drive/My Drive/Colab Notebooks/Random forest/X_train3.p','rb'))
X_test3 = pickle.load(open('drive/My Drive/Colab Notebooks/Random forest/X_test3.p','rb'))
```

In [51]:

```
%env JOBLIB_TEMP_FOLDER=/tmp
```

```
env: JOBLIB_TEMP_FOLDER=/tmp
```

In [110]:

```
X_train3.shape,X_test3.shape
```

Out[110]:

```
((70000, 200), (30000, 200))
```

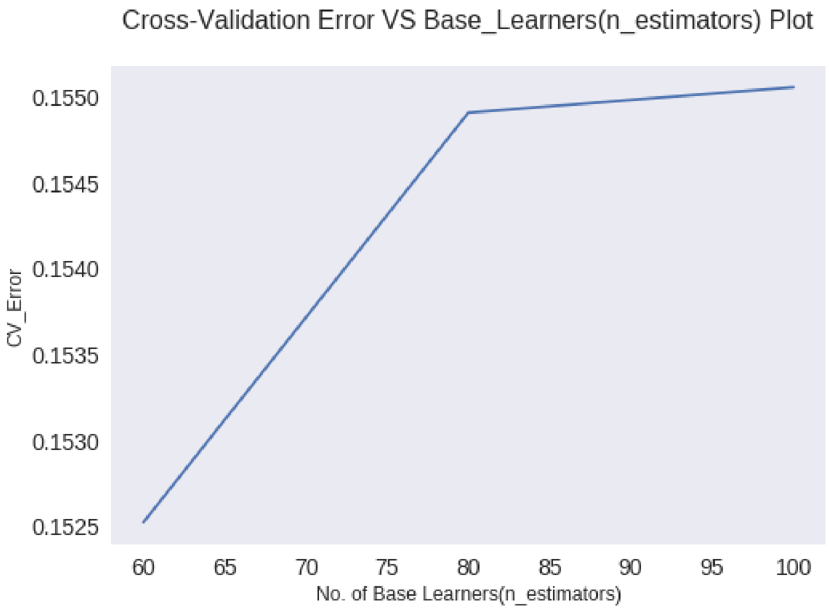
In [52]:

```
tscv = TimeSeriesSplit(n_splits=3)
param_grid = {'n_estimators':[60,80,100]}
rf=RandomForestClassifier(oob_score=True,class_weight='balanced')
gsv = GridSearchCV(rf,param_grid,cv=tscv,scoring="f1_weighted",n_jobs=-1,pre_dispatch=2)
gsv.fit(X_train,y_train)

print("Best HyperParameter: ",gsv.best_params_)
print("Best f1: %.2f%%"%(gsv.best_score_*100))
```

```
Best HyperParameter:  {'n_estimators': 60}
Best f1: 84.75%
```

In [53]:

```
base_learners=[60,80,100]
cv_errors = [1-i for i in gsv.cv_results_['mean_test_score']]
# plotting Cross-Validation Error vs Base learners graph
plt.plot(base_learners, cv_errors)
plt.xlabel('No. of Base Learners(n_estimators)',size=12)
plt.ylabel('CV_Error',size=12)
plt.title('Cross-Validation Error VS Base_Learners(n_estimators) Plot\n',size=16)
plt.grid()
plt.show()
```



In [114]:

```
model3 =RandomForestClassifier(oob_score=True,n_estimators=14,class_weight='balanced')
model3.fit(X_train3,y_train)
y_pred3 = model3.predict(X_train3)
model3
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:453: UserWarning: Some inputs do
not have OOB scores. This probably means too few trees were used to compute any reliable oob estim
ates.
  warn("Some inputs do not have OOB scores. "
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:458: RuntimeWarning: invalid val
ue encountered in true_divide
  predictions[k].sum(axis=1)[:, np.newaxis])
```

Out[114]:

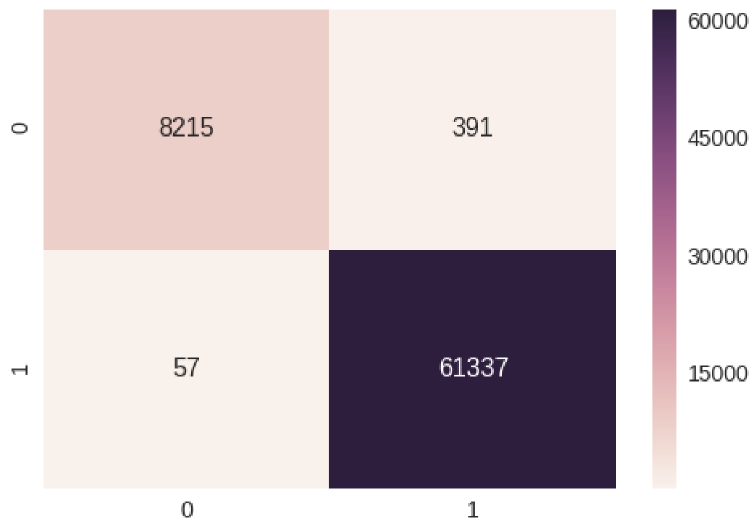
```
RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=None, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=14, n_jobs=1, oob_score=True, random_state=None,
            verbose=0, warm_start=False)
```

In [116]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import seaborn as sns

print("Accuracy on the set: %0.3f%%"%(accuracy_score(y_train, y_pred3)*100))
print("F1-Score on the set: %0.3f%%"%(f1_score(y_train, y_pred3,average='weighted')*100))
print("Precision_score on test set: %0.3f%%"%(precision_score(y_train, y_pred3)*100))
print("Recall_score on test set: %0.3f%%"%(recall_score(y_train, y_pred3)*100))
result = confusion_matrix(y_train,y_pred3)
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
```

```
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print(result)
sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Accuracy on the set: 99.360%
F1-Score on the set: 99.355%
Precision_score on test set: 99.367%
Recall_score on test set: 99.907%
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[ 8215   391]
 [   57 61337]]
F1-Score on the set: 99.636%
```



In [118]:

```
y_pred3 = model3.predict(X_test3)
model3
```

Out[118]:

```
RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=None, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=14, n_jobs=1, oob_score=True, random_state=None,
            verbose=0, warm_start=False)
```

In [119]:

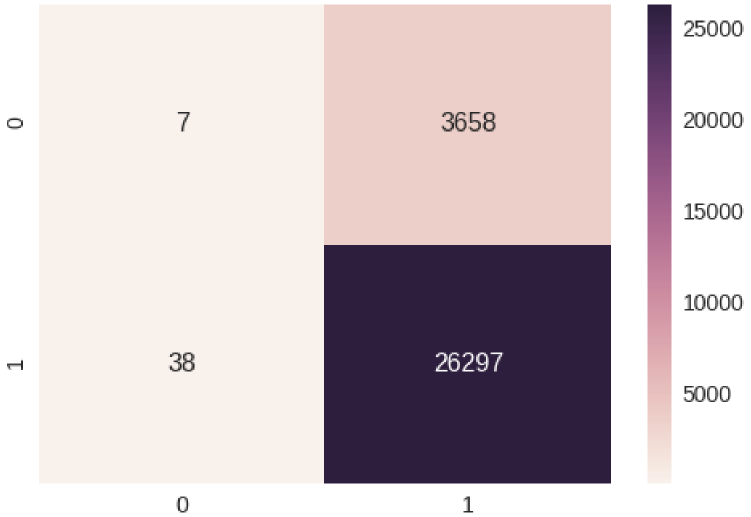
```
print("Accuracy on the set: %0.3f%%"%(accuracy_score(y_test, y_pred3)*100))
print("F1-Score on the set: %0.3f%%"%(f1_score(y_test, y_pred3,average='weighted')*100))
print("Precision_score on test set: %0.3f%%"%(precision_score(y_test, y_pred3)*100))
print("Recall_score on test set: %0.3f%%"%(recall_score(y_test, y_pred3)*100))

print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
result = confusion_matrix(y_test,y_pred3)
print(result)
sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Accuracy on the set: 87.680%
F1-Score on the set: 82.066%
Precision_score on test set: 87.788%
Recall_score on test set: 99.856%
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[     7  3658]
```

```
 [   38 26297]]
```

Out[119]:

```
&lt;matplotlib.axes._subplots.AxesSubplot at 0x7f8cffca72b0&gt;
```



AVG TF-IDF W2V

In [54]:

```
y = final['Score'].iloc[:100000]
x = final['CleanedText'].iloc[:100000]
x.shape,y.shape
```

Out[54]:

```
((100000,), (100000,))
```

In [0]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

X_tra, X_tes, y_train, y_test = train_test_split(x.values,y.values,test_size=0.3,random_state=0)
```

In [0]:

```
sent_of_train=[]
for sent in X_tra:
    sent_of_train.append(sent.split())

sent_of_test=[]
for sent in X_tes:
    sent_of_test.append(sent.split())
```

In [57]:

```
#word to vector
from gensim.models import Word2Vec
w2v_model=Word2Vec(sent_of_train,min_count=3,size=200, workers=4)# words which occurs 3 times; 500 dimensions

w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 3 times ",len(w2v_words))
```

```
number of words that occured minimum 3 times  14075
```

In [0]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
m = TfidfVectorizer()
tf_idf_matrix = m.fit_transform(X_tra)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(m.get_feature_names(), list(m.idf_)))
```

In [59]:

```
from tqdm import tqdm
import numpy as np

# TF-IDF weighted Word2Vec
tfidf_feat = m.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(sent_of_train): # for each review/sentence
    sent_vec = np.zeros(200) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████| 70000/70000 [01:53<00:00, 617.25it/s]
```

In [60]:

```
tfidf_sent_vectors1 = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(sent_of_test): # for each review/sentence
    sent_vec = np.zeros(200) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors1.append(sent_vec)
    row += 1
```

```
100%|██████████| 30000/30000 [00:50<00:00, 599.52it/s]
```

In [61]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import TimeSeriesSplit

# Data-preprocessing: Standardizing the data
sc = StandardScaler(with_mean = False)
X_train4 = sc.fit_transform(tfidf_sent_vectors)
X_test4 = sc.transform(tfidf_sent_vectors1)

X_train4.shape,X_test4.shape
```

Out[61]:

```
((70000, 200), (30000, 200))
```

In [0]:

```
pickle.dump(X_train4, open('X_train4.p', 'wb'))
pickle.dump(X_test4, open('X_test4.p', 'wb'))
```

In [84]:

```
##############################################################3
X_train4 = pickle.load(open('drive/My Drive/Colab Notebooks/Random forest/X_test4.p','rb'))
X_test4 = pickle.load(open('drive/My Drive/Colab Notebooks/Random forest/X_test4.p','rb'))
X_train4.shape,X_test4.shape
```

Out[84]:

```
((30000, 200), (30000, 200))
```

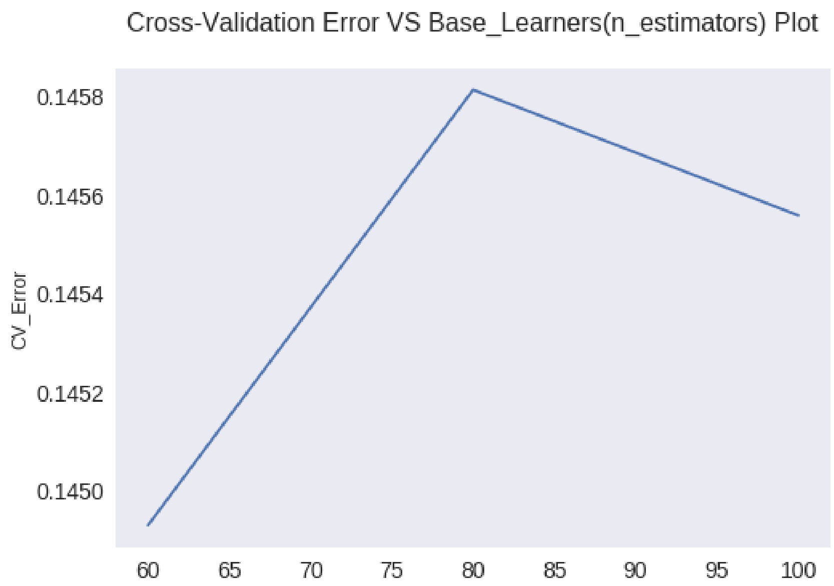
In [62]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import make_scorer
from sklearn.metrics import f1_score
from sklearn.model_selection import RandomizedSearchCV

tscv = TimeSeriesSplit(n_splits=3)
param_grid = {'n_estimators':[60,80,100]}
rf=RandomForestClassifier(oob_score=True,class_weight="balanced")
gsv = GridSearchCV(rf,param_grid,cv=tscv,scoring="f1_weighted",n_jobs=-1,pre_dispatch=2)
gsv.fit(X_train4,y_train)

print("Best HyperParameter: ",gsv.best_params_)
print("Best f1: %.2f%%"%(gsv.best_score_*100))
```

```
Best HyperParameter:  {'n_estimators': 60}
Best f1: 85.51%
```

In [63]:

```
cv_errors = [1-i for i in gsv.cv_results_['mean_test_score']]

base_learners=[60,80,100]
# plotting Cross-Validation Error vs Base learners graph
plt.plot(base_learners, cv_errors)
plt.xlabel('No. of Base Learners(n_estimators)',size=12)
plt.ylabel('CV_Error',size=12)
plt.title('Cross-Validation Error VS Base_Learners(n_estimators) Plot\n',size=16)
plt.grid()
plt.show()
```



No. of Base Learners(n_estimators)

In [64]:

```
rf=RandomForestClassifier(n_estimators=60,oob_score=True,class_weight="balanced")
rf.fit(X_train4,y_train)
y_pred = rf.predict(X_train4)
rf
```

Out[64]:

```
RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=None, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=60, n_jobs=1, oob_score=True, random_state=None,
            verbose=0, warm_start=False)
```

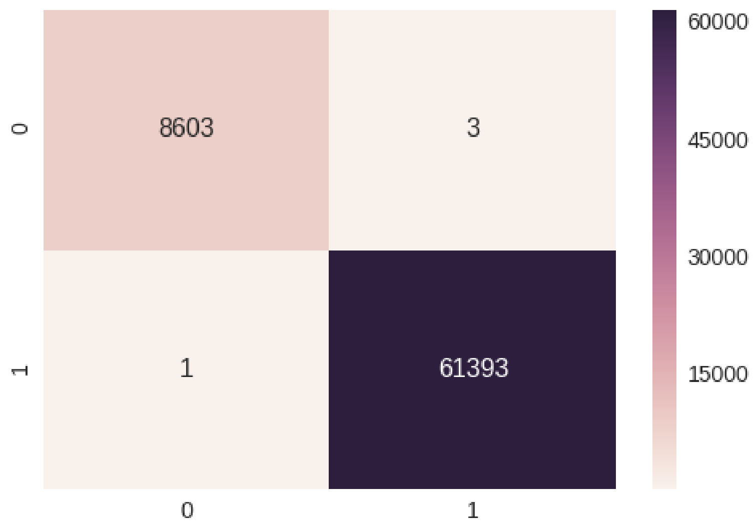
In [65]:

```
print("Accuracy on the set: %0.3f%%"%(accuracy_score(y_train, y_pred)*100))
print("F1-Score on the set: %0.3f%%"%(f1_score(y_train, y_pred,average='weighted')*100))
print("Precision_score on test set: %0.3f%%"%(precision_score(y_train, y_pred)*100))
print("Recall_score on test set: %0.3f%%"%(recall_score(y_train, y_pred)*100))
result = confusion_matrix(y_train,y_pred)
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print(result)
sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Accuracy on the set: 99.994%
F1-Score on the set: 99.994%
Precision_score on test set: 99.995%
Recall_score on test set: 99.998%
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[ 8603     3]
 [    1 61393]]
```

Out[65]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4b7619a160>
```



In [126]:

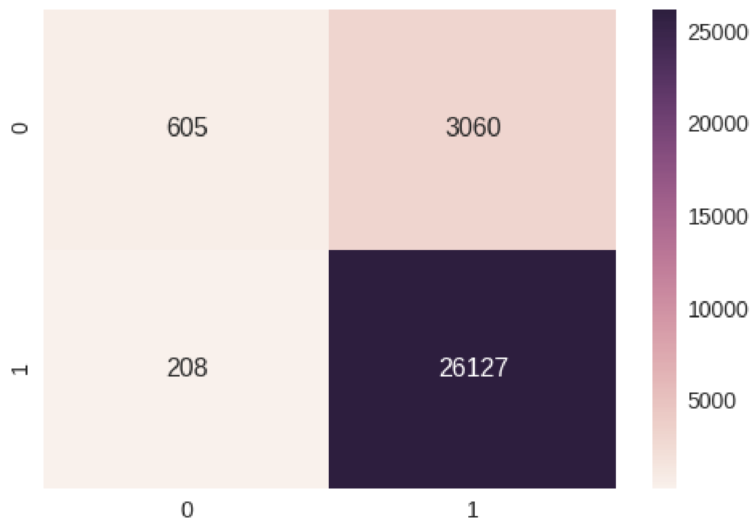
```
y_pred = rf.predict(X_test4)
print("Accuracy on the set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("F1-Score on the set: %0.3f%%"%(f1_score(y_test, y_pred)*100))
print("Precision_score on test set: %0.3f%%"%(precision_score(y_test, y_pred)*100))
print("Recall_score on test set: %0.3f%%"%(recall_score(y_test, y_pred)*100))
```

```
result = confusion_matrix(y_test,y_pred)
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print(result)
sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Accuracy on the set: 89.107%
F1-Score on the set: 94.114%
Precision_score on test set: 89.516%
Recall_score on test set: 99.210%
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[  605  3060]
 [  208 26127]]
```

Out[126]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8cffcc54a8>
```



In [1]:

```
import pandas as pd
df = pd.read_csv('rf.csv')
df
```

Out[1]:

	Unnamed: 0	F1 score of Train	F1 score of Test	Hyperparameter
0	BOW	99.9%	85.83%	60
1	TFIDF	99.99%	84.17%	60
2	AVG-W2V	99.35%	82.06%	60
3	TFIDF-W2V	99.99%	94.11%	60

Random forest of TFiDF-W2V has best performance.